The mapping package for the Alice Dimuon Arm Spectrometer

D. Guez, I. Hrivnacova, M. Maccormick

INSTITUTE
IPN Orsay
91406 ORSAY Cedex
France

Internal NOTE
ALICE Refrence Number
ALICE-INT-2003-024 version 1.0
Institute reference number
[-]
date of last change
2nd July 2003

The mapping package for the ALICE Dimuon Arm Spectrometer

D. Guez, I. Hřivnáčová, M. MacCormick
IPN Orsay, 91406 ORSAY, France

Abstract

The ALICE Dimuon Arm Spectrometer mapping package is described in this report. After an introduction describing the relevant detector and electronics parameters, the contents of the software package will be described. It is currently implemented for station 1, has been used successfully in test-beam prototype experiments and also in the general ALICE simulation program AliRoot. A guide to the program architecture, showing how to run and test the code is given.

IPNO-DR-03-05

The mapping package for the ALICE Dimuon Arm Spectrometer

D. Guez, I. Hřivnáčová, M. MacCormick

(2nd July 2003)

1 Introduction

In the dimuon arm spectrometer [1], over the 10 tracking multi-wire proportional chambers, there are approximately 1.2 million individual charge coding channels. These channels are implemented in the form of pads which are etched onto the surfaces of the two cathode planes of each detector. The way in which they are distributed over station 1 (which regroups the first two tracking chambers) will be looked at in detail in the following paragraphs.

Each channel is connected to one unique pad segment of a cathode pad plane, each pad having a unique spatial location. All channels are read out individually via parallel, multiplexed lines and are stored event-by-event in a 32 bit word containing the charge measured and its local electronics ID tag number. The output lines are funnelled into two successive levels of so-called "concentrator" boards, where they are concactenated and attributed a second ID tag. Consequently, a charge measured on a pad at position (x,y,z) is coded under two unique ID numbers, specified by the system designers. In the following, it is the first electronics ID tag that is of interest. The second one is treated by a separate acquisition code.

The overall mapping consists of :

- finding the electronics ID tag for a pad when given its spatial coordinates;
- determining spatial coordinates from an electronics ID tag;
- locating the nearest pad neighbours indifferently, starting from the electronics ID or pad position;
- providing the means to correlate the charge measurements on opposing cathode planes, and
- doing all of this in an efficient, user friendly way.

The elementary problem description is quite straightforward, and constitutes a standard experimental situation. As long as the key to decoding the electronics information is given, and the experimental layout is known, then there is no particular problem. In the dimuon spectrometer it is the volume of data, and its organisation into manageable entities which is of interest.

In order to explain this in greater detail, the overall layout of the different electronics entities will be presented along with their relationship to the geometrical pad layout scheme. The different kinds of mapping that are used in current detector work are given, and clearly define the different levels of interest of different working groups. Finally, the architecture of the program itself is described along with a description of how to run and test it.

Geometry, electronics, symmetries and terminology

The general layout of the 4 quadrants of the segmented cathode MWPC's that make up the two tracking chambers of station 1 is shown in figure 1 [1]. Station 2 is also shown, it has a different internal geometry, but is built on the same model as station 1. Each quadrant has two active planes, that measure the spatial coordinates $(x,y)_{mm}$ of incident particles. The spectrometer's dipole magnetic field curves trajectories in the "y" direction and is commonly called the "bending" coordinate. The "non-bending" coordinate therefore corresponds to the "x" direction. The bending cathode plane has a segmented pad geometry that gives a high resolution y-coordinate measurement, and its layout is different from that of the the non-bending plane which provides the lower resolution x-coordinate

There are three different zones defined on each cathode plane, going from smaller segments in the inner radii that will allow to resolve the higher density of impact points

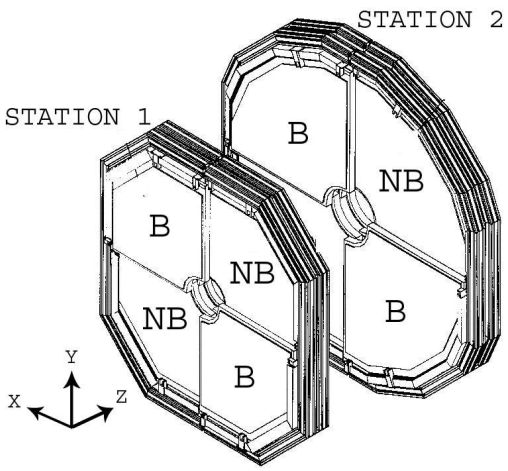


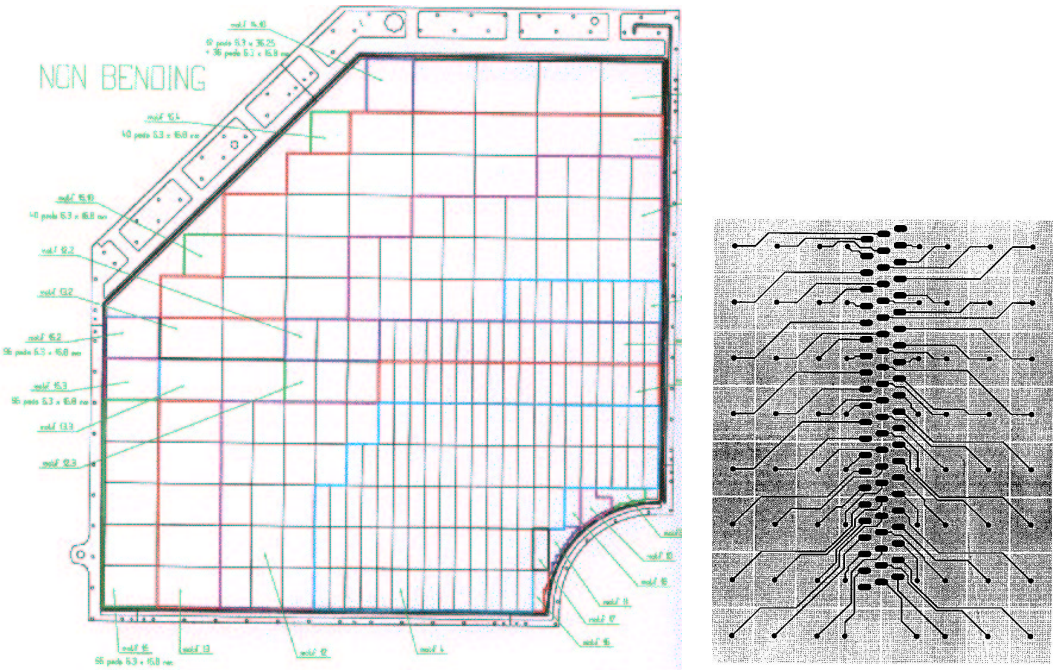
Figure 1: A view of the layout of the chambers of stations 1 and 2. Each station is composed of two independent chambers, and each chamber is composed of four quadrants.

in this region, to the larger segments in the outer zones. An example of the overall layout for a non-bending plane is shown in figure 2(a). The bending and non-bending template is repeated for each of the quadrants, and the final chamber layout is composed of 4 identical quarter chambers that are mounted together with a simple reflection symmetry. The coordinate system of each quadrant is a reflection in the common axis of adjacent elements.

The layout of the cathode surface is organized into groups of 64 elementary pad blocks called “motifs". The motifs' wiring schemes, are based on repetitive designs, the aim being to simplify the overall detector design and usage by having as much symmetry as possible. An example of a typical motif layout is shown in figure 2(b). The 8x8 rectangular grid represents the pad surfaces that measure the charge signal, and the lines are the feed-through copper rails that transmit the signal to the 66 point flexible Kapton^{TM} connector that would be soldered on to the 3 rows of points visible in the design. Located at the other end of the Kapton connector is an 80 point connector that allows to plug the flexible Kapton into the 64-channel front-end data analysis card. This connector is the so-called "Berg" connector referred to in the mapping package. A close-up of the two sides of a front-end data analysis card, with the major components highlighted, can be seen in figure 3(a).

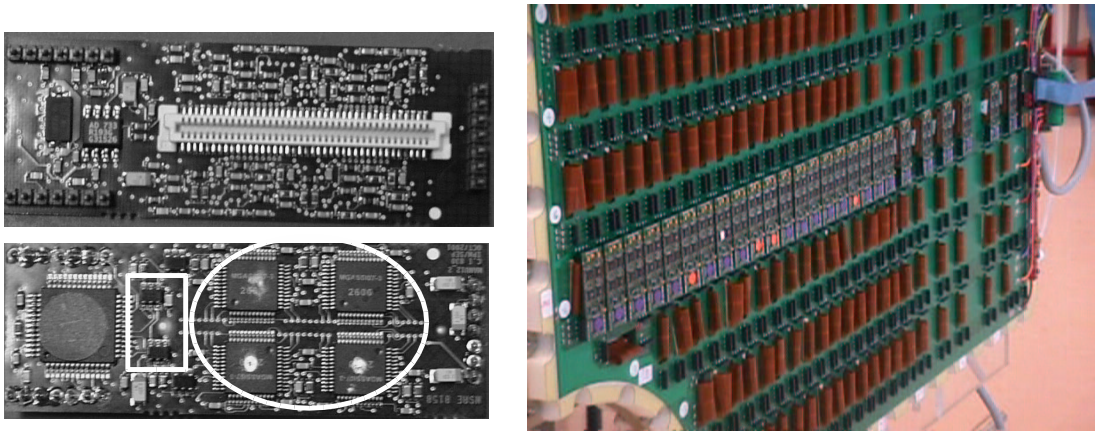
A photograph of a quadrant is shown in figure 3(b) and the regular layout of the readout electronics can be seen. The system, similar in appearence for both cathode planes, is composed of lines of electronics cards, plugged into a common bus, that regroups up to a maximum of 26 front-end cards. Each front-end card allows to measure and code 64 individual pads (ie., one motif). It is the digital output from these cards that transit in the buses lines that run from left to right in the photograph 3(b).

These outputs are then input, in sets of 5, to a sequence of DSP's implanted on front-end and concentrator electronics boards. These boards regroup the data from the different detector buses (there are a total of 13 buses per quarter chamber regrouping the 30,000 elementary channels), concactenate them and, eventually communicate them through the ALICE DDL (Detector Data Link) and on to the common part of the ALICE acquisition system [2].



(a) (b)

Figure 2: (a) Layout of the major elements on the non-bending plane. Three distinct zones, corresponding to three different elementary pad sizes can be seen. Each rectangle symbolises the outline of the individuals motifs. (b) An example of a typical motif showing a set of 64 pads and the wiring scheme to the central connector is visible.



(a) (b)

Figure 3: (a) A close-up of a 64 channel data analysis card. The upper image show the back-side of the card. The 80 point “Berg" connector that links up with the pads via the flexible Kapton connector can be seen. This is therefore the entry point for the charge signals. The lower image shows the front view, and the main charge analysis components are identified. The 4 GASSIPLEX/Manas chips are highlighted by the oval shape, and the white rectangle shows the position of the two ADC chips. The card deals with a total of 64 channels, corresponding to one whole motif. (b) A photograph of a quarter chamber. The regular electronics readout scheme can be seen on the outside of the detector. One readout bus is implemented with its front-end cards, each card has a multiplexed 64 channel digital output that contain the coded charge measurement from 64 individual pads. The bus is read from left to right in the photograph.

The different tasks

Different kinds of mapping are required, depending on the task in hand. In the mapping software, this has been taken into account. The path followed by a single charge measurement, in going from a unique pad at position (x,y) (in mm) on a cathode plane to its final storage position the general ALICE event architecture, is shown symbolically in table 1. In this simplified view, there are different interfaces with each unique pad signal being attributed a fixed transit channel at each level. In order to retain the memory of this trip, and to provide a unique address, the datum picks up two ID tags along the way. The first ID tag, attributed at the output of the 64 channel analysis card, identifies the channels used for each pad signal on the analysis card. This ID is held in a 17-bit word that allows to identify the GASSIPLEX/Manas and ADC channels as well as the card ID itself. The second tag is attributed during the data concatenation on the "CROCUS" data concentrator boards. Its coding and decoding is treated by a separate code. It is only through decoding both ID tags that the original pad position of a measurement can be determined in the full spectrometer layout. However, for an isolated quadrant, the first level ID contains enough information for identifying a unique pad location, and it is this ID that is of interest here.

Charge measurement	$\longrightarrow (x,y)_{mm/pad\ units}$		Data Taking	
HARDWIRED	IN : 66 PT CONNECTOR OUT : 80 PT CONNECTOR	flexible Kapton^{TM} connector	⇓ ⇓ ⇓ ⇓	
ELECTRONICS ID TAG #1	16 CHANNEL GASSIPLEX/MANAS ANALYSIS CHIP 32 CHANNEL 12 BIT ADC	64 channel analysis card		⇑
ELECTRONICS ID TAG #2	SUCCESSIVE DATA CONCENTRATOR CARDS	"CROCUS" system		⇑ ⇑ ⇑ ⇑
Charge storage	→ ALICE DDL			Data Analysis

Table 1: Summary of the different elements through which a single charge measurement transits. Only the main electronics functions are highlighted. The mapping at each interface must be accessible for debugging and monitoring. The initial and final points are used for the event reconstruction in the online, offline and simulation work.

The arrows in the end columns of table 1 show the usual direction of interest, depending on whether data is being taken or analysed. However, the end point is not the same for all users.

In table 2 the minimum requirements for different types of work are given. For the sake of simplicity, the most common tasks are listed. The cross marks the type of mapping level that is usually required in each of these situations. In the physics simulation world, one does not care too much about the order in which the electronics cards are plugged in, since the main aim is to simulate and manipulate the information on the detectors' cathode planes. The important point here, is that functions, such as determining the nearest neighbour, use the same interface as in the offline data analysis. In this way the same analysis code can be used in both cases, hence rendering the overall effort more efficient.

On the other hand, the detailed detector simulation, which uses real experimental data as an input, requires a different level of mapping. Each individual channel is associated with its full set of experimental characteristics, ie. pedestal, noise and threshold levels, calibration factor and status flag. The mapping of the electronics numbering system

is essential in coordinating and associating this information in the correct way. Every effort has been made to ensure that the detailed detector simulation code truly corresponds to the experimental, data taking environment.

	pad $(x,y)_{mm}$	Elecronics coding filters	Final electronics ID
physics simulation	X		
detector simulation	X	X	
offline analysis	X	X	X
online monitoring		X	X
laboratory tests		X	

Table 2: The most frequent tasks that use the same mapping are listed in the left-hand column. The column headings are the major mapping levels, as annotated in table 1, while the cross marks the minimum need for each task. All tasks can use all the mapping levels if so desired.

In test-beam activities, where online monitoring is used, the first questions asked are generally to do with the state of the electronics readout and detector tuning. It's therefore more appropriate to be able to efficiently view the response from individual electronics modules. The intermediate electronics mapping levels are more important in this case.

Although all tasks can use all the mapping levels if so desired, it is in the offline analysis, where the full mapping is used. In this case, data are delivered with their full electronics ID, the associated charge may be submitted to various cuts and calibrations before being united with its unique physical pad location. Event reconstruction is built up from different sources of information for example, mapping information, calibration data, and dead channel masks.

The mapping package was developed in order to have a generic mapping code that could be used indifferently in full scale simulation, test-beam data taking or offline data analysis. It can also be used in laboratory electronics testing, in situations where only the electronics chain, minus the detector is generally used. As will be shown in the following, all of this can be done for partially or fully implemented detectors, using exactly the same code, the user only needs to provide some data files.

2 Program architecture

The mapping package exploits object-oriented technology and is written in C++. It is based on the ROOT framework [3] and has been developed for the ALICE experiment [4], it consequently follows the ALICE C++ Coding Conventions [6].

The various classes have been structured into categories that reflect the physical design of stations 1 and 2 (as previously shown in figures 1 and 2). These categories are as follows :

basic - common elementary classes, interfaces;
motif - classes related to the motif;
sector - classes related to the composition of sector;
plane - classes related to the plane, and
graphics - classes for visualization.

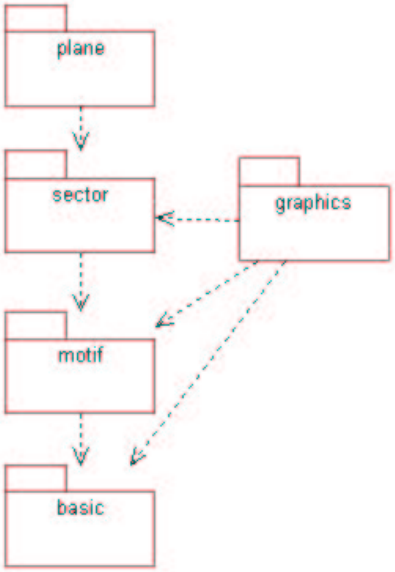


Figure 4: Class categories

As can be seen from the class diagram structure in figure 4, the category dependencies are unidirectional.

The detector description uses the following entity definitions :

pad - smallest element;
motif - repeatedly placed ensemble of pads (in most cases rectangular) with given characteristics;
row segment - a segment of a row of motifs containing motifs of the same type;
row- row of motifs composed of row segments;
subzone - region of the plane with the same motifs;
zone - region of the plane with pads of the same dimension;
sector - plane quadrant;
there is a requirement for a constant pad size in one direction, and
plane - whole plane, composed of four sector positions.

The relationship between these entities and the physical components of the detector can be deduced from figures 1 and 2.

For some areas of the detector description, there are special cases that need to be taken into account. This is done through the definition of abstract base classes, the derived

classes are then implemented in both standard and specific cases. In the case of motif, the standard motif is composed of pads of the same dimensions while a special motif can be composed of pads with varying dimensions. In the case of row segment, the standard row segment is composed of the same, rectangular motifs, while the special row segment is composed of unique, not rectangular motifs that can go beyond the standard row area.

Besides the entities that make up the plane composition, there are also logical entities that allow to navigate between the pads :

segmentation - provides functions for retrieving pads, starting from given characteristics (ie., position, location, and indices) and for finding pad neighbours (ie., up, down, right, and left), and

pad iterator - which provides the means to iterate over the pads.

In the folowing paragraphs the key classes for each category will be described.

2.1 Basic classes

The category overview is presented in the class diagram of figure 5. This category includes all the common elementary classes, the interfaces and utility classes.

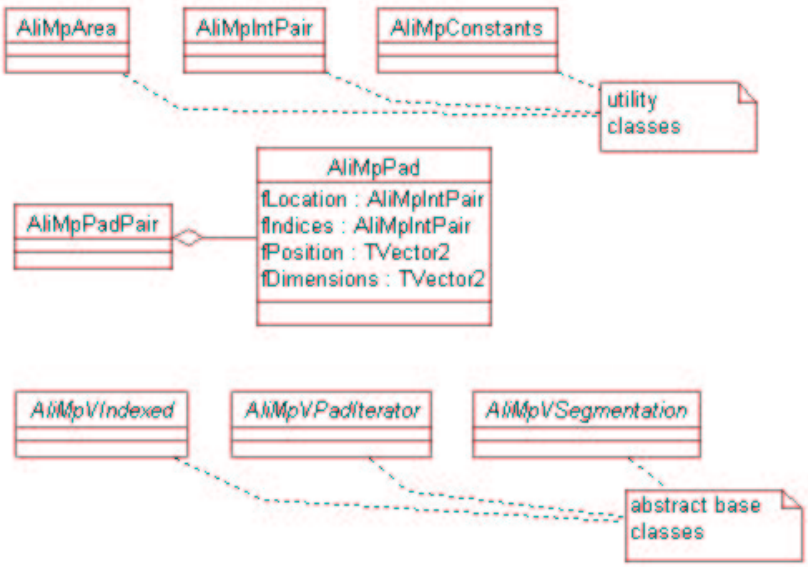


Figure 5: Basic class diagram.

AliMpPad is the most frequently used elementary class, it defines the pad object with the following characteristics :

position - position of the pad (x,y) in the global coordinate system;

dimensions - half length of the pad size in x and y;

indices - the number of the pad in the row, and the column in the plane (ix,iy);

location - motif position identifier plus the "Berg" connector number of the pad

The interfaces *AliMpVSegmentation* and *AliMpVPadIterator* are introduced for the logical entities segmentation and pad iterator, they will be discussed further in section 2.5. The *AliMpVIndexed* interface defines the common properties for all structural elements with indices.

2.2 Motif classes

The category overview is presented in the class diagrams of figures 6 and 7.

The motif in the cathode plane is represented by the motif object which is defined as being a composition of the information concerning the pad size (or pad sizes in the special motif case) and the motif type. The motif type contains, among other characteristics,

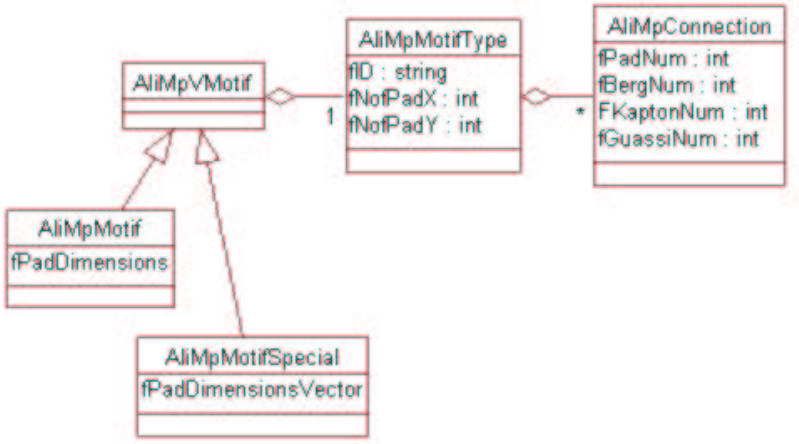


Figure 6: Motif class diagram.

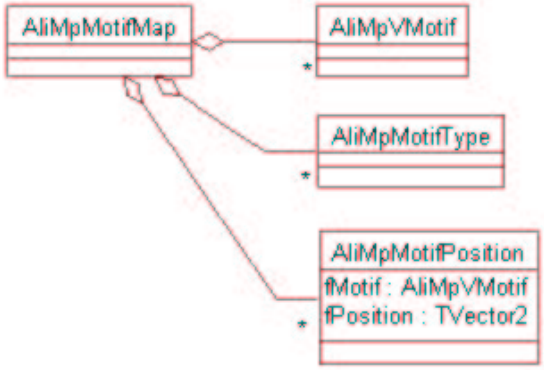


Figure 7: Motif map class diagram.

a map of all its connections. The connection describes the local electronics channel/pad characteristics :

Berg connector number;

Kapton connector number, and

GASSIPLEX/Manas channel number.

The relationships between the key classes *AliMpVMotif*, *AliMpMotifType*, and *AliMpConnection* are shown in the class diagram of figure 6. The physical objects are shown in figure 2(b) and the photographs of figure 3.

The placing of the motif on the cathode plane is represented by the motif position object. Each motif position has a unique integer identifier, which is specified in the input data and is identical to the one used during data acquisition.

In order to provide a fast access to the motif, motif types and, motif position objects, the maps between the object identifiers (string or integer) and the pointers to these object are built. These maps are contained in the motif map object defined by the *AliMpMotifMap* class (shown in figure 7).

2.3 Sector classes

The category overview is presented in the class diagrams of figures 8 and 9. To

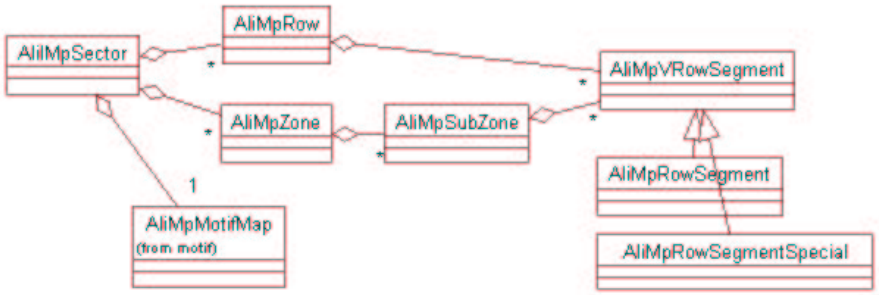


Figure 8: Sector class diagram

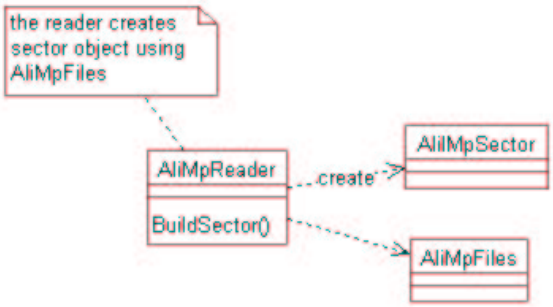


Figure 9: Sector reader class diagram.

facilitate the definition of the sector layout, the row segment entity was introduced. The standard row segment is defined as being a row of motifs of the same type; the mapping package requires that the motif position identifiers that belonging to the same row segment are sequentially increasing or decreasing. The special row segment case will be described later in this section.

The key class of this category is the sector class, named *AliMpSector*, that represents a physical quadrant of the plane, and can be compared to figure 2(a). The sector is composed of row segments which are organised into two complementary logical structures, namely:

- in rows - this logical structure follows the physical design of the plane; the row is composed of row segments having the same y coordinate value.
- in zones and subzones - subzones are composed of the row segments which have motifs of the same type and pad dimensions; zone is a set of subzones which all have the same pad dimensions.

The relevant classes, and their relationships, are shown in the class diagram of figure 8.

The sector is built up from the ASCII data files provided by a user and read by the sector reader object, defined by the *AliMpReader* class. More will be said about data files in section 3.1. All file names and paths are defined in the *AliMpFiles* class. The sector reader classes are shown in figure 9.

Special zones

In the inner part of the non-bending quadrant of station 1 the motifs have a unique and irregular pad layout (see figure 2(a)). As well as this, the y-coordinate can go beyond the standard row size (the standard layout is 8 pads high). This area, which has been called a special zone in the mapping, is then decomposed into special row segment objects, defined in the *AliMpRowSegmentSpecial* class. This class follows the row segment interface *AliMpVRowSegment*, as shown in figure 8. A special row segment is defined via pad rows, defined in the *AliMpPadRow* class which are composed of pad row segments, defined in the *AliMpPadRowSegment* class. The pad row segments then define which pads belong to the same motif.

The current implementation supports the inner special zones and is now being extended to include the special zones that exist in the outer edges of station 2.

2.4 Plane classes

The category overview is presented in the class diagram of figure 10. The plane class

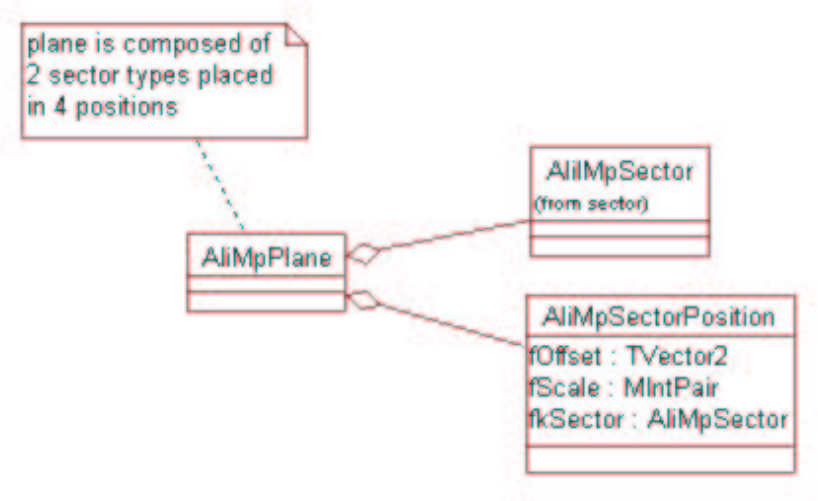


Figure 10: Plane class diagram.

AliMpPlane, defines a plane as being a composition of four sector positions, as defined by the *AliMpSectorPosition* class. The position of each sector is specified through 2D translations and reflections.

The current plane definition is defined in only 2-dimensions, the 3D transformation is not yet available. This functionality will be provided in future in order to accomodate the information provided by the external geometrical alignment procedure [7]. The plane class can also be easily generalized so that any number of sector positions can be instantiated. This is a requirement for stations 3, 4 and 5 where ladders of slats (as opposed to quadrants) are used to build these larger detectors.

2.5 Segmentation and Pad Iterator classes

The segmentation and pad iterator are logical entities that allow to navigate through the pads. As both these entities can be applied at different levels (ie., motif, sector and plane), the segmentation and pad iterator classes are present in almost all categories.

For both the segmentation and pad iterator instances, the abstract base classes are first defined. The segmentation concrete classes are then implemented for sector and plane

elements. The iterator concrete classes are implemented to iterate over various structural elements (such as motif, row segment, sector, etc...) as well as over a rectangular area in the plane, or in the sector.

The abstract factory pattern [8] is applied to the segmentation and pad iterator classes. The segmentation concrete classes implement the function for creating the iterator. The iterator is made in such a way that is the appropriate one for the associated component (ie., motif, row segment, sector, etc...). The efficient searching algorithms in the implementation of the segmentation and pad iterator classes were developed using the standard C++ library.

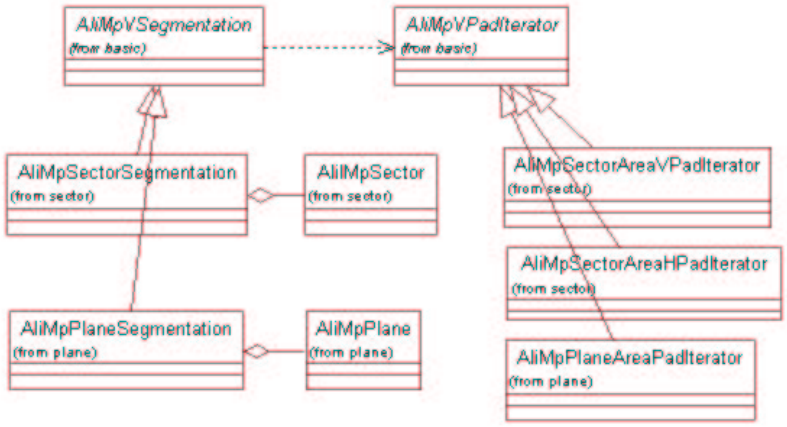


Figure 11: Segmentation and pad iterator (class diagram).

The category overview is presented in the class diagram of figure 11.

2.6 Graphics classes

The category overview is presented in the class diagram of figure 12.

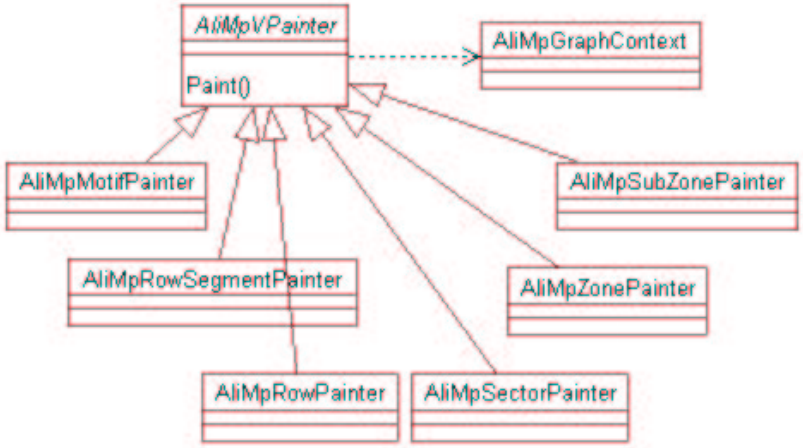


Figure 12: Graphics class diagram.

For most entities in the mapping package that represent real life objects, there is an associated graphics class that allows it to be drawn on the screen. *AliMpVPainter* is the base class for all such classes. These concrete “painter" classes essentially overload the

standard ROOT Paint() method with the appropriate code. In all, there are six classes that derive from *AliMpVPainter* that allow to draw the motif, row segment, row, subzone, zone and sector objects.

As well as the Paint() method, each concrete class also has a Draw() function, as required by ROOT, which can take an input argument. If no argument is provided, the object will draw itself on the screen. However, if an argument is supplied the object can be drawn, not only by the painter itself, but also by instanciating a painter for each component of the object. A full detailed example is provided in the mapping package documentation.

3 How to use and test

In order to use the mapping package the user needs to provide the data files describing the detector. The front-end mapping classes (segmentation and pad iterator) can then be used directly from a ROOT UI (terminal or macro), or from the user client code. The description of the mapping data files and an example of how to use the mapping in a client code will be given in the following sections.

3.1 Data files

All input parameters that define the plane topology, as well as the electronics properties, have to be specified in the set of ASCI data files :

zones.dat - definition of the sector layout;

zones_special.dat - special zones definitions;

motifX.dat - motif type characteristics and electronics wiring;

(X represents a motif type identifier), and

motifPosX.dat - mapping between indices in the motifX.dat file and local (i,j) on the motif.

The detailed desription of the files' format is provided in the mapping package documentation.

3.2 Use of mapping in client code

The mapping package is currently used in the TestBeam ToolBox [9] online monitoring program, in the test-beam data analysis and in the AliRoot simulation program [5]. The front-end classes for the client code are the segmentation and pad iterator classes. In example 3, we show how the user can instantiate the plane object, create its segmentation and use the iterator to print the properties of all pads in a specified rectangle of the plane.

3.3 Using the graphics classes

The graphics classes are used at different stages of the project. When verifying the mapping code itself, errors can be easily pinpointed by viewing the positions of the individual elements. In the client code used in prototype test experiments, being able to easily superimpose the beam impact points on the mapped elements has greatly speeded up the setting up process. As well as this, should any error occur during data taking, the exact location of the problem can be quickly identified visually via its electronics ID, hence making the monitoring process more efficient. A typical monitor image is shown in figure 13. Finally, in the latest developments in the data analysis code, the graphics classes have been implemented so as to have a visual comparison of the zone where data was measured and the reconstructed impact point. Once again, this facilitates the event by event debugging process.

3.4 Test suite

When developing the code, a test ROOT macro was written for each new part and as a result there are around 15 test macros. The testing procedure has been simplified by writing a Perl script which executes all the macros automatically, and compares the output with the reference output.

Graphics output from the test macros is demonstrated in figures 14 and 15. The first figure, figure 14, shows the sector elements, namely the zones, subzones, row segments, and motifs. The layout of each element is drawn in the canvases. In figure 15 the motifs, with their motif position identifiers, are shown.

```
AliMpPlane* plane
   = AliMpPlane::Create(kBendingPlane);
                  // The plane object is created in this function;
                  // the AliMpReader class is used to read the
                  // input data files and build the sectors with
                  // all their structural elements

AliMpVSegmentation* segmentation
   = new AliMpPlaneSegmentation(plane);
                  // Create the segmentation associated with the plane

AliMpArea area(TVector2(10., 20.), TVector2(20., 20.));
                  // Define the area over which to iterate;
                  // the first argument is the position of the center,
                  // the second argument are the half-lengths

AliMpVPadIterator* it = segmentation -> CreateIterator(area);
                  // Creates the iterator over the area

for (it -> First(); ! it -> IsDone(); it -> Next()) {
                  // Loop over the pads

      cout << iterator << CurrentItem() << endl;
                  // Print the properties of the current pad
}
```

Table 3: An example of a user code

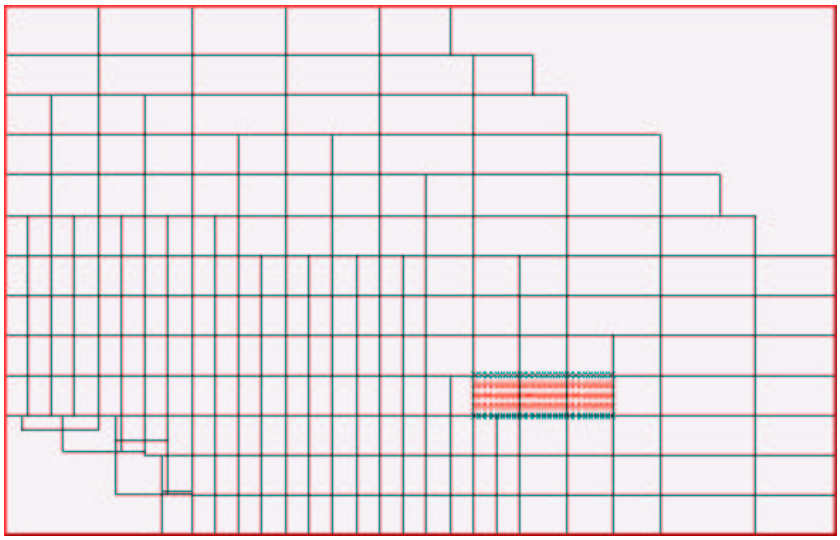


Figure 13: An example of an online monitoring image. The overall detector mapping can be seen with the zone of accumulated beam impact points marked by the small crosses.

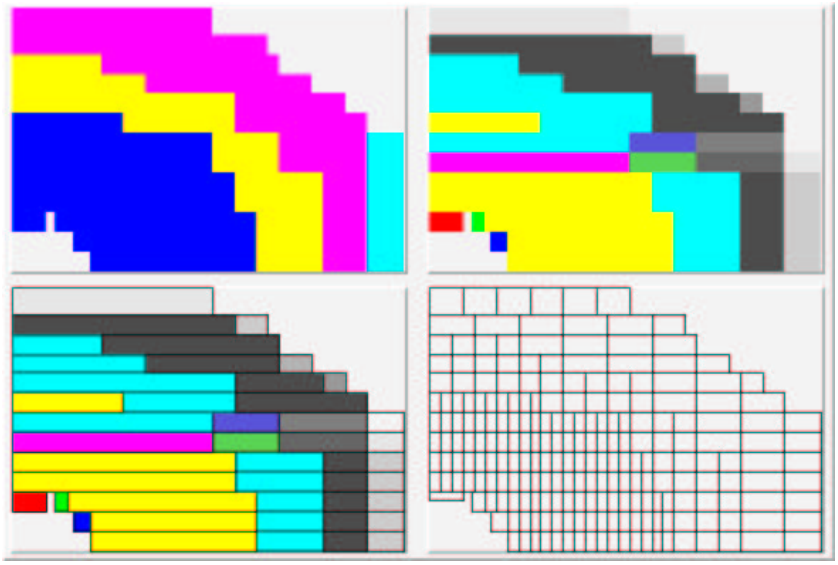


Figure 14: The sector elements: zones, subzones, row segments and motifs as drawn from a test ROOT macro.

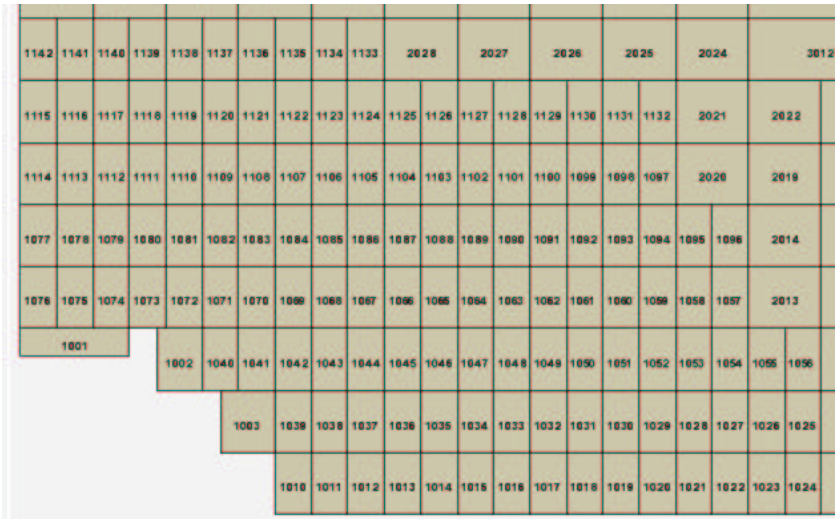


Figure 15: Motifs with motif position identifiers as drawn from a test ROOT macro.

4 Distribution

The mapping package has been included in AliRoot version 3.09, in the MUON library. In order to obtain a copy, and to install AliRoot, the procedure described at [5] should be followed.

Locally, a stand-alone version is used by physicists and engeneers for the purposes of testing electronics and test-beam data analysis. The installation procedure is described in the mapping package documentation.

5 Conclusions

The mapping package has been successfully implemented on station 1 of the ALICE Dimuon Arm Spectrometer for test-beam online monitoring and data analysis, and also in the detailed AliRoot detector simulation program.

Applying the object-oriented methodology allowed to define the technical parameters in a manageable, user-friendly way and will facilitate any future data file modifications. The graphics functions, based on the ROOT system, makes the mapping package interesting for a bench test user.

References

[1] "ALICE Dimuon Forward Spectrometer TDR", CERN/LHCC 99-22, 13 August 1999.
[2] ALICE DATE, CERN ALICE Data Acquisition Group.
[3] http://root.cern.ch
[4] "ALICE - Technical Proposal for A Large Ion Collider Experiment at the CERN LHC", CERN/LHC/95-71, December 1995.
[5] http://AliSoft.cern.ch/offline
[6] http://AliSoft.cern.ch/offline/codingconv.html
[7] "Addendum 1 to ALICE TDR 5", Chapter 4 and references therein, CERN/LHCC 2000-046, December 2000.
[8] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Software. Addison Wesley, 1995.
[9] D. Guez and M. MacCormick, et al.: TestBeam ToolBox Reference Guide; ALICE-INT-2001-45, IPNO DR-02-001.